АКАДЕМИЈА ТЕХНИЧКО-УМЕТНИЧКИХ СТРУКОВНИХ СТУДИЈА БЕОГРАД

ОДСЕК ВИСОКА ШКОЛА ЕЛЕКТРОТЕХНИКЕ И РАЧУНАРСТВА

**Михић Јован**

**НАСЛОВ РАДА**

**- завршни рад -**



Београд, месец 20\_\_.

Кандидат: **Презиме Име**

Број индекса: **117/18**

Студијски програм: **Нове Рачунарске Технологије**

Тема: **XXZXXXXXXXXXXXXXXXXXXXX**

Основни задаци:

**1. Xxxxxxxxxxxxx**

**2. Xxxxxxxxxxxxx**

**3. Xxxxxxxxxxxxx**

Ментор:

Београд, месец 20\_\_ годинe.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

м/др Зоран Ћировић, проф. ВИШЕР

Резиме:

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

**Кључне речи:** : Xxxxxxxxxxxxxxxxxxxxxxxxxxxxx

ABSTRACT:

Xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

**Key words**: Xxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Садржај

[1. УВОД 1](#_Toc160183064)

[2. Методологија 2](#_Toc160183065)

[2.1. Истраживачки приступ 2](#_Toc160183066)

[2.2. Одабир технологија 2](#_Toc160183067)

[2.3. Методологија развоја 2](#_Toc160183068)

[2.4. Алати и оквири 3](#_Toc160183069)

[2.4.1. Развојно окружење 3](#_Toc160183070)

[2.4.2. База података и серверска инфраструктура 3](#_Toc160183071)

[2.4.3. Корисничка апликација 4](#_Toc160183072)

[2.5. Планирање пројекта 5](#_Toc160183073)

[3. Дизајн и архитектура 6](#_Toc160183074)

[3.1. Клијентска апликација 6](#_Toc160183075)

[3.1.1. *Vue.js* оквир 6](#_Toc160183076)

[3.2. Серверска апликација 8](#_Toc160183077)

[*Node.js* 8](#_Toc160183078)

[*Express.js* 8](#_Toc160183079)

[4. Имлепентација 9](#_Toc160183080)

[4.1. Серверска апликација 9](#_Toc160183081)

[4.1.1. Модели базе података 9](#_Toc160183082)

[4.1.2. Рутирање 11](#_Toc160183083)

[4.1.3. Контролери 12](#_Toc160183084)

[4.1.4. Аутентификација корисника 13](#_Toc160183085)

[4.1.5. Middleware функције 15](#_Toc160183086)

[4.2. Клијентска апликација 16](#_Toc160183087)

[4.2.1. Рутирање 16](#_Toc160183088)

[4.2.2. Приказ страница 18](#_Toc160183089)

[4.2.3. Складиштење податка 19](#_Toc160183090)

[5. ЗАКЉУЧАК 23](#_Toc160183091)

[6. ИНДЕКС ПОЈМОВА 24](#_Toc160183092)

[7. ЛИТЕРАТУРА 25](#_Toc160183093)

[8. Прилози 26](#_Toc160183094)

[9. изјава о академскoј честитости 27](#_Toc160183095)

# УВОД

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

# Методологија

## Истраживачки приступ

Приступ: експлоративни.

Опис: Истраживање различитих технолошких опција за развој *MEVN* стек (*stack*) апликације за онлајн трговину стрипова. Анализа постојећих алата, технологија и библиотека за израду корисничке и серверске веб апликације. Истраживање предности, мана и најбољих пракси сваке технологије како би се утврдило најбоље решење за израду апликације за онлајн трговину.

Методе: Преглед литературе, анализа предности и мана, истраживање онлајн ресурса и тестирање прототипова.

Резултати: Идентификација *MEVN* стека као најоптималнијег решења за израду апликације за онлајн трговину, због његове флексибилности, скалабилности и перформанси.

## Одабир технологија

Одабран је *MongoDB* за реализацију базе података због његове скалабилности и брзине.

*Vue.js* је одабранза израду корисничког интерфејса због његове једноставности и реактивности.

*Node.js* и *Express.js* су одабрани за израду серверске апликације због њихове ефикасности, једноставности и приступу *Node Package Manager*-у (*NPM*) који омогућава лак увоз додатних алата и библиотека у апликацију.

## Методологија развоја

С обзиром на то да је овај пројекат развијен од стране једне особе, а не тима дизајнера, програмера и тестера, одабран је „агилни“ приступ развоју пројекта. Ова методологија је одабрана због своје флексибилности.

Агилни приступ омогућује итеративни развој апликације, у ком се апликација лако мења услед промењених циљева/планова који настају због непредвиђених проблема и потешкоћа.

За једну особу је тешко да предвиди све могуће проблеме, зато је овакав приступ идеалан за овај пројекат.

## Алати и оквири

### Развојно окружење

***Visual Studio Code***: Развојно окружење одабрано за овај пројекат је *Visual Studio Code,* бивајући да је то најкоришћеније окружење за развој са *JavaScript*-ом и због одличне интеграције са системом за верзионирање *Git* и интегрисаним терминалом, који олакшава развој апликације, који уклањања потребу за отварањем нових прозора за терминал и навигирање до радног директоријума у терминалу*.*

***Git, GitHub***: Коришћен је систем за верзионирање *Git*, који је био од виталног значаја са обзиром на то да се неколико пута у току развоја кôд морао враћати на претходну верзију. Такође, систем за верзионирање је омогућио да се развој апликације врши са било које машине, приступањем *GitHub-у,* онлајн платформи за *Git*.

### База података и серверска инфраструктура

***MongoDB Atlas****: MongoDB Atlas* је *cloud* сервис коришћен за хостинг *MongoDB* базе података. *MongoDB Atlas* омогућава скалабилност, високу доступност и поузданост базе података.

***MongoDB Compass****: MongoDB Compass* је графички интерфејс за базу података који омогућава приказ и лаку измену података у бази података.

***Mongoose:*** *Mongoose* је алатка за моделовање *MongoDB* објеката, *ODM* (*Object Document Mapper),* дизајнирана да ради у асинхроном окружењу. Знатно олакшава рад са *MongoDB-*ем.

**Express.js**: *Express.js* је коришћен као веб оквир за израду серверске стране апликације. *Express.js* је омогућио брз и ефикасан развој серверских рута и *API*-ја.

***Node Package Manager****: Node Package Manager* је менаџер пакета коришћен за увоз додатних постојећих алата и библиотека у апликацију.

***JWT:*** *JWT* ***(****JSON Web Token)* је технологија за аутентификацију корисника издавањем „токена“ који сваки корисник чува при сваком *HTTP* захтеву га шаље као доказ да је он заиста тај корисник за којег се представља. Коришћена је за реализовање аутентификације корисника.

***Bcrypt.js:*** *Bcrypt.js* је алатка за хеширање (*hashing*). Коришћена је за побољшање сигурности корисничких налога јер омогућава да се у бази чувају хеширане лозинке, а не отворени текст.

***Multer:*** *Multer je middleware* за руковање са *multipart/form-data*. Коришћен је да би омогућио постављање слика стрипова на сервер.

***Morgan:*** *Morgan* је *middleware* за исписивање *HTTP* захтева послатих серверу у његову конзолу. Коришћена је ради лакшег проналажења грешака у коду.

***Nodemon:*** *Nodemon* јеалатка која омогућује да се *Express.js* сервер аутоматски поново покреће сваки пут када се сачува промена у коду. Коришћена је јер знатно олакшава израду сервера.

### Корисничка апликација

***Vuetify****:* *Vuetfiy* је библиотека *Vue.js* компоненти за графички интерфејс. Омогућава да се израде лепи графички интерфејси са минималним *CSS*-ом, што знатно убрзава израду веб апликације.

***Axios:*** *Axios* je *HTTP* клијент базиран на обећањима (*promise based*)*.* Може се извршавати и на северу и у претраживачу. Коришћен је за лакше слање *HTTP* захтева серверској апликацији.

***Pinia:*** *Pinia* је складиште података за *Vue.js* апликације. Коришћена је за кеширање (*Caching*) података добијених од сервера у клијентској апликацији.

***Vue Router:*** *Vue Router* је рутер за *Vue.js* једностране апликације. Омогућава мапирање одређених компоненти за различите руте претраживача. Коришћен је да приказује одређене странице за задате *URL*-ове.

## Планирање пројекта

Циљеви:

* Приказ свих стрипова корисницима са филтрирањем по едицији стрипа и могућност сортирања.
* Приказ детаљних информација стрипа.
* Регистровање нових корисника са важећим *e-mail*-ом и одабраном лозинком.
* Пријављивање (*login*-овање) регистрованих корисника.
* Додавање одабраних стрипова у корисничку „корпу“
* Евентуална измена садржаја своје „корпе“ од стране корисника.
* Слање поруџбине са садржајем „корпе“.
* Могућност прегледа својих претходних поруџбина и њихово стање ( „послато“ или „није послато“).
* Реализација администраторског налога који има овлашћење за додавање нових производа и приступ прегледу поруџбина свих корисника и означавање њих као „послате“ након њихове обраде.
* Могућност мењања својства постојећих стрипова, од стране администратора, са тим да се промене над стриповима пресликавају у „корпи“ корисника, уколико оне садрже тај стрип (нпр. ако се промени цена стрипа, цена стрипа у корпи ће бити промењена).

Ресурси: Бивајући да је ово студентски пројекат, буџет за израду овог пројекта је непостојећи, тако да су сви алати/технологије/хостинг бесплатни или отвореног кода.

Временски оквири: Једини временски оквир у овом пројекту је тај да се мора завршити у року од годину дана након пријаве теме.

# Дизајн и архитектура

Модерне веб апликације развијају се помоћу разних различитих технологија (програмски језици, оквири (*Framework*), базе података, фронт енд и бек енд алатке и *API* (*Application Programming Interface*)). Ове технологије не морају бити писане у истом програмском језику, нити се морају извршавати на истој машини, модуларне су и могу се заменити неком другом технологијом уколико настане потреба за тим. Ове одабране технологије заједно чине скуп који се назива стек (*stack*).

Одабрани стек за овај пројекат је *MongoDB, Express.js, Vue.js, Node.js*, познат као „**MEVN**“ стек (***M****ongoDB,* ***E****xpress.js,* ***V****ue.js* ***N****ode.js*).

Пројекат се састоји од две независне и одвојене апликације, замишљене да раде на различитим серверима. Једна је клијентска апликација, која је јавно доступна корисницима, а друга серверска апликација која комуницира са базом података и испоручује податке или их мења на захтев клијентске апликације.

## Клијентска апликација

Клијентска апликација је интерфејс са којим корисници интерагују, приступају јој помоћу интернет претраживача. Њен главни циљ је да омогући корисницима да интерагују са подацима на серверу путем „*user-friendly*” корисничког интерфејса. Реализована је помоћу *Vue.js* оквира за програмски језик „*JavaScript*” и *Vuetify* библиотеке компонената за . *Vue.js.*

### *Vue.js* оквир

„Vue.js“ je фронт енд *model-view-viewmodel* *JavaScript* библиотека отвореног кода за израду корисничких интерфејса и једностраних апликација (*single-page application*).

Направљен 2014. године, од стране Евана Јуа, по узору на *AngularJS* оквир, са циљем да узме све што „ваља“ у *Angular*-у а изостави све непотребне и конфузне делове. Као такав, *Vue.js* је убрзо постао један од најпопуларнијих оквира за *JavaScript*.

*Vue.js* користи компоненте (*components*) које проширују основне *HTML* елементе и омогућавају да се направе нови елементи (компоненте). Једном направљена компонента се може користити у коду неограничено много пута и тиме смањује редудантност кода. Апликација је сведена на мноштво мањих логичких целина (компонената) што знатно олакшава одржавање апликације.

* + - 1. ***Vue.js* Шаблони**

Имплементирана је шаблонска синтакса по угледу на *HTML* која омогућује везивање рендерованог *DOM-a* (*Data Object Model)* за вредности података у *Vue* инстанци. *Vuе* компилује шаблоне у свом виртуелном *DOM-у.* Виртуелни *DOM* омогућује да се компоненте рендерују у меморији пре него што се ажурирају у интернет претраживачу.

На овај начин, у комбинацији са *Vue-*овим системом реактивности, *Vue* може да израчуна минималан број комоненти за поновно рендеровање и тако примени минималну количину промена у *DOM-*у. Тиме се постижу знатно боље перформансе.

* + - 1. **Систем реактивности**

*Vue.js* имплементира систем реактивности који користи обичне *JavaScript* објекте и оптимизовано поновно рендеровање. Свака компонента прати своје реактивне зависности (*dependecies)* током свог рендеровања, и тако систем зна тачно када да поново рендерује коју компоненту, не трошећи време и ресурсе рендерујући компоненте које се неће променити*.*

* + - 1. **Транзиције**

Омогућено је да се са лакоћом додају анимације за било коју компоненту када је она створена, ажурирана или уклоњена из *DOM*-а.

* + - 1. **Рутирање**

Бивајући да једностране апликације, за разлику од традиционалних веб сајтова, имају само једну страну, потребан је систем рутирања који ће променити оно што је рендеровано на страници у зависности од задатог *URL*-а у претраживачу и тиме дати привид различитих страна. Рутер је потребан да би омогућио навигацију веб сајтом путем линкова и чување веб историје.

Рутер је доступан након инсталирања пакета „*vue-router“* у *Vue.js* апликацију.

* + - 1. ***Pinia***

*Pinia* је библиотека за складиштење података за *Vue.js* и оквир за управљање стања (*state management framework)* у *Vue.js-*у. Замишљена као унапређена верзија *Vuex-*a (који је првобитно био коришћен као складиште података за овај пројекат, али је на крају замењен *Pini-*ом), за разлику од свог претходника, дизајнирана је да буде потпуно модуларна и не укључује мутације над стањима. Пружа централизовано складиште са уграђеним механизмима за чување, ажурирање и „дохватање“ стања.

### *Vuetify* Библиотека

Велика пажња је посвећења корисничком искуству. *Vuetify*, библиотека компонената корисничког интерфејса коришћена у пројекту, је заснована на *Google*-овим *Material Design* спецификацијама (те спецификације су осмислили *Google*-ови дизајнери пратећи најбоље праксе дизајна корисничких интерфејса), и као такав *Vuetify* гарантује да ће се, самим његовим коришћењем, кориснички интерфејс придржавати тим спецификацијама.

*Vuetify* даје опцију додавања разних прелазних анимација на све *Vue.js* компоненте, те анимације су коришћене при свакој промени странице и дају лепше корисничко искуство.

### Дизајн апликације

#### Почетна страница

Почетна страница коју корисници виде је страница са приказом свих стрипова које могу купити. Стрипови су приказани као слике насловне стране стрипа са едицијом насловом и бројем.

Сваки стрип има дугме за додавање у корпу, чијим кликом корисник може брзо додати жељени стрип у корпу. Уколико корисник није пријављен на свој налог, онемогућено му је додавање стрипа у корпу. Поред овог дугмета налази се и дугме за детаљније информације о стрипу које води корисника на страницу са детаљнијим информацијама о том стрипу (кликом на саму слику стрипа се такође може приступити овој страници).

#### Страница са детаљнијим информацијама о стрипу

На овој страници кориснику је приказана слика стрипа на коју може кликнути и добити већи приказ слике у виду модалног дијалога. Поред слике стоји едиција, наслов, број, целовити опис стрипа и цена стрипа.

Након ових информација стоје поље за унос жељене количине стрипа и дугме за додавање у корпу (које је онемогућено ако корисник није пријављен).

Испод свега овога налази се карусел сличних производа са три стране по три стрипа (прва три су увек из исте едиције, ако их има толико у бази података, а осталих 6 су насумични стрипови). Кликом на неког од њих отвара се страница са детаљним информацијама тог стрипа.

#### Страница за пријављивање и мени за пријављене корисника

У горњем десном углу апликације постоји дугме за пријаву корисника, кликом на њега корисник је одведен на страницу за пријаву. Страница за пријаву се састоји од два поља за унос текста, једно за корисников *e-mail*, друго за његову лозинку. Испод ових поља постоји линк ка страници за регистрацију у случају да корисник нема регистрован налог.

Након пријављивања на свој налог, кориснику се са десне стране (уместо дугмета за пријаву) појављује мени на извлачење, путем којег може приступити својој „корпи“, поруџбинама или одјавити се са свог налога. Уколико је корисник администратор појављују му се додатне опције (у том менију) за додавање нових производа и преглед свих поруџбина.

#### Страница корпе

Кликом на дугме за корпу кориснику се приказује страница корпе. У случају да је корпа празна, кориснику је дат линк ка почетној страници и порука која му говори да тамо може додати стрипове у своју корпу. Након што корисник дода стрипове у корпу овде ће му бити приказани у виду листе чији су елементи ти стрипови које је додао.

Сваки елемент листе има слику стрипа (ради лаке визуелне идентификације производа), едицију, наслов, број, цену, количину и производ множења цене и количине. Такође постоје дугмићи за брисање стрипа из корпе и мењање количине у случају да се корисник предомислио и жели наручити друкчију количину.

На дну ове листе налази се укупна цена свих производа и дугме за прављење поруџбине.

#### Страница поруџбине

Кликом на дугме за поруџбину корисник је одведен на страницу за поруџбину. Ова страница се састоји од форме за унос података за испоруку (име поручиоца, презиме поручиоца, контакт телефон, улица, број куће/стана, град и поштански број).

Уколико корисник жели да унесе адресу наплате друкчију од адресе испоруке, може то урадити тако што ће одштиклирати опцију за коришћење исте адресе за испоруку и наплату. Овом акцијом се појављују поља за адресу наплате.

Корисник има могућност да штиклира опцију за чување података које је унео, како не би морао следећи пут да их опет уноси. Када направи поруџбину подаци ће му бити сачувани у бази података и сваким следећим отварањем ове странице подаци ће бити већ попуњени, наравно он их може изменити ако то жели и чак опет сачувати те нове податке.

Испод ових опција налази се листа свих производа које је наручио са њиховим ценама и њиховом укупном ценом, што даје потврду кориснику онога чега је наручио пре него што финализира своју поруџбину.

Испод ове листе је дугме које ће направити ову поруџбину. Кликом на ово дугме унети подаци, заједно са садржајем корпе се шаљу серверу и чувају у бази података. Потом се садржај корпе празни, како би корисник могао да направи нову поруџбину.

## Серверска апликација

Серверска апликација је апликација са којом клијентска апликација интерагује. Клијент шаље *HTTP* захтеве серверу, а сервер на основу тих захтева шаље тражене ресурсе или мења податке у бази података.

Реализована је као ***RESTful API*** (*representational state transfer)*. *RESTful API* је интерфејс који омогућује пренос података између две машине на стандардизован начин, без обзира на технологије коришћене.

Принцип рада је следећи: Када клијент пошаље захтев за неки ресурс (на пример захтев за листу свих стрипова) сервер „дохвати“ жељени ресурс из базе података и пошаље га у виду *JSON(JavaScript Object Notation)* стринга, затим клијент добијени стринг парсира у *JSON* објекат и са њиме прави приказ жељеног ресурса кориснику. Такође је могуће послати захтев за измену ресурса или креирање нове инстанце ресурса, у том случају сервер шаље *JSON* стринг у ком је порука о успешној промени/креирању ресурса или поруку о грешци уколико промена или креирање нису били успешни.

Велика предност *RESTful API-ева* је та што су подаци који су враћени клијенту у облику *JSON* стринга. Тиме се осигурава да је могуће користити те податке на било ком другом систему, независно од програмског језика у ком је клијент писан. Клијенту се само прослеђују **репрезентације** података (у виду *JSON* стрингова), не сами подаци.

Сервер апликација у овом пројекту је реализована у *Node.js* окружењу.

За израду самог *API-a* је коришћен *Express.js* оквир,који је *de facto* стандардни оквир за израду *API-a* у *Node.js* окружењу

За базу података је одабрана *MongoDB* технологија, имплементирана у виду *mongoose* пакета.

### *Node.js*

*Node.js* је *JavaScript* систем за извршавање кода (*runtime environment)* отвореног кода који извршава *JavaScript* код ван интернет претраживача. Другим речима, омогућује девелоперима да пишу серверски код у *JavaScript-у.*

Садржи моћан менаџер пакета (*node package manager)*, помоћу којег се јако лако у апликацију додају друге апликације (пакети).

### *Express.js*

Express.js је бек енд *JavaScript* оквир отвореног кода за израду веб апликација и *RESTfull API-ева у Node.js* окружењу.

# Имлепентација

## Серверска апликација

### Модели базе података

Модели у бази података у овом пројекту су следећи:

1. „order.js“ – Овај модел представља поруџбину коју прави корисник. Састоји се од идентификатора корисника који је направио поруџбину, листе стрипова које је поручио, имена корисника, презимена корисника, адресе испоруке, адресе наплате, датума поручивања и стања поруџбине.
2. „product.js“ – Овај модел представља производ, то јест стрип. Састоји се од едиције, наслова, броја издања, описа, цене и *URL*-а слике стрипа.
3. „user.js“ – Овај модел представља корисника. Састоји се од *e-mail*-а, лозинке, корисничког имена, улоге (корисник или администратор), корпе (листа идентификатора стрипова и њихових количина) и података за поруџбину (подаци које корисник уписује када прави поруџбину које може да сачува, како их не би опет уписивао).

Пример *orders.js* модела:

* const addressSchema = mongoose.Schema({
* city: { type: String, required: true },
* street: { type: String, required: true },
* houseNumber: { type: String, required: true },
* postalCode: { type: String, required: true },
* });
* const orderSchema = new Schema({
* userId: {
* type: Schema.Types.ObjectId,
* required: true,
* },
* products: [
* {
* productId: {
* type: Schema.Types.ObjectId,
* required: true,
* ref: "Product",
* },
* quantity: {
* type: Number,
* required: true,
* },
* },
* ],
* firstName: {
* type: String,
* required: true,
* },
* lastName: {
* type: String,
* required: true,
* },
* shippingAddress: {
* type: addressSchema,
* // required: true,
* },
* billingAddress: {
* type: addressSchema,
* // required: true,
* },
* orderDate: {
* type: Date,
* default: new Date(),
* },
* isShipped: {
* type: Boolean,
* default: false,
* },
* });
* module.exports = mongoose.model("Order", orderSchema);

### Рутирање

Принцип рада сервера је такав да му се шаљу *HTTP* захтеви на одговарајуће *URL-*ове. Да би примио и одговорио на те захтеве мора да има дефинисане руте.

Сваки *URL*  мора да почиње са адресом сервера, затим портом на ком ради, затим мора да садржи „*/api/v1/“*. Оваква структура је имплементирана да би се у будућности, евентуално, могла направити нова верзија *API-*а („*/api/v2“, „/api/v3“*…) без да се уклони већ постојећа функционалност (што је у овом студентском пројекту потпуно непотребно, али је генерално добра пракса).

Након овог дела у *URL-*у следи једна од рута дефинисаних у рутеру. Уколико се пошаље непостојећа рута клијент ће добити *error* са поруком „*not found*“.

Када сервер прими захтев за неку руту, он примењује *middleware* функције које су дефинисане за ту руту (на пример *middleware* функција *authenticate* која проверава да ли је корисник онај који тврди да јесте), па позива одговарајући контролер.

Контролер затим обрађује захтев и враћа одговор клијенту који је послао захтев.

Напомена: Технички није било потребно одвајати контролере у посебне фајлове, али је добра пракса, јер чини код много прегледнијим, поготово у случајевима већих пројеката, који имају јако велик број котнролера.

Дефинисана су три фајла за рутирање:

1. „orders.js“ – Садржи све руте везане за поруџбине
2. „products.js“ – Садржи све руте везане за продукте то јест стрипове
3. „user.js“ – Садржи све руте везане за корисничке налоге (корисничка „корпа“, регистрација новог корисника, пријављивање корисника…)

Пример *user.js* рутера:

const express = require("express");

const router = express.Router();

const authenticate = require("../middleware/authenticate");

const UserController = require("../controllers/user");

router.get("/", UserController.user\_get\_all);

router.post("/signup", UserController.user\_signup);

router.post("/login", UserController.user\_login);

router.get("/cart", authenticate, UserController.user\_get\_cart);

router.post("/cart", authenticate, UserController.user\_add\_to\_cart);

router.delete("/cart/:id", authenticate, UserController.user\_remove\_from\_cart);

router.patch("/cart", authenticate, UserController.user\_edit\_cart\_item);

router.get(

  "/checkout-info",

  authenticate,

  UserController.user\_get\_checkout\_info

);

router.post(

  "/checkout-info",

  authenticate,

  UserController.user\_save\_checkout\_info

);

module.exports = router;

### Контролери

Као што је већ пре напоменуто, свака рута позива одговарајућу функцију у контролеру. За сваки рутер направљен је одговарајући конторлер. Контролер је у суштини само *JavaScript* фајл који експортује све своје функције. Он се импортује у одговарајућем рутеру, и тако рутер има приступ свим тим функцијама, које позива при добијеном *HTTP* захтеву.

Пример *products\_get\_all* функције из *products.js* контролера:

exports.products\_get\_all = async (req, res, next) => {

  const editions = await Product.getEditions();

  Product.find()

    .select("edition title issue description price imageUrl")

    .exec()

    .then((products) => {

      const response = {

        count: products.length,

        products: products.map((p) => {

          return {

            ...p.\_doc,

          };

        }),

        editions,

      };

      res.status(200).json({

        message: "Fetched products",

        response,

      });

    })

    .catch((err) => {

      console.log(err);

      res.status(500).json({

        error: err,

      });

    });

};

Принцип рада ове функције је да се над модел *Product* позива *mongoose* функција *find()* (ова функција, без аргумената, враћа све постојеће *MongoDB* документе тог модела), након чега се користи *then() catch()* синтакса која је потребна с обзиром на то да се ради са асинхроном функцијом.

У *then()* блоку се позива анонимна функција која за аргумент *products* узима добијене документе које је пронашла *find()* функција.

Конструише се константа *response* која има вредност објекта са кључевима *count* (укупан број пронађених стрипова), *products* (низ пронађених стрипова са свим њиховим подацима) и *editions* (низ свих постојећих едиција стрипова, користи се да би клијент апликација знала које све едиције постоје када даје кориснику опцију да филтрира по едицијама).

Затим се овај објекат *response* шаље као одговор на *HTTP* захтев уз поруку *„Fetched products“* (да би корисник овог *API*-a имао увид у то шта му је послато) са статус кодом 200, што значи да је захтев успешно обрађен.

Уколико из неког разлога дође до грешке у *then()* блоку извршава се *catch()* блок у ком се у конзоли (ради лакшег проналажења грешака) исписује грешка која се десила и као одговор на *HTTP* захтев шаље се иста та грешка, јер корисник *API*-а нема приступ конзоли сервера и мора му се на неки начин дати до знања који проблем је у питању. Одговор има статус код 500, што кориснику говори да је дошло до интерне грешке на серверској страни.

### Аутентификација корисника

Како би смо обезбедили да сваки корисник има своју персоналну корпу и своје персоналне поруџбине, потребно је да постоје кориснички налози.

У овом пројекту то је реализовано уз помоћ *JWT*-а (*JSON Web Token*). Принцип рада *JWT-*а је следећи:

* Приликом пријављивања корисника, сервер уз помоћ тајне која је само њему позната (*JWT Secret*) генерише „токен“ који је уникатан за тог корисника (садржи његов јединствени идентификатор).
* Токен служи као потврда да је корисник заиста тај за ког се представља да јесте (као нека врста „личне карте“ за корисника). Битно је напоменути да ако се неко „дочепа“ туђег токена, могуће је да га користи за зле намере и да приступа подацима тог корисника, те је пожељно да се с времена на време освежавају токени (то јест да се ограничи трајање токена и да се пошаље нови токен након његовог истека).
* Пријављени корисник сада може користити свој токен при слању *HTTP* захтева тако што ће у заглавље сваког захтева, за који је потребна аутентификација, ставити кључ „Authorization“ са вредношћу „Bearer“, затим размаком па кодираним токеном.
* Сада сервер при сваком захтеву за који је потребна аутентификација, проверава валидност овог токена и обрађује захтев уколико је он валидан.

Пример контролера за пријављивање:

exports.user\_login = (req, res, next) => {

  User.findOne({ email: req.body.email })

    .exec()

    .then((user) => {

      // console.log(user);

      if (!user) {

        return res.status(401).json({

          message: "Authentication failed",

        });

      }

      bcrypt.compare(req.body.password, user.password, (err, result) => {

        if (err) {

          return res.status(401).json({

            message: "Authentication failed",

          });

        }

        if (result) {

          const token = jwt.sign(

            {

              \_id: user.\_id,

            },

            process.env.JWT\_KEY

          );

          return res.status(200).json({

            message: "Authentication successful",

            token: token,

            \_id: user.\_id,

            name: user.name,

            email: user.email,

            role: user.role,

            checkoutInfo: user.checkoutInfo,

          });

        }

        return res.status(401).json({

          message: "Authentication failed",

        });

      });

    })

    .catch((err) => {

      console.log(err);

      res.status(500).json({

        error: err,

      });

    });

};

Над моделом *User* се примењује *mongoose* функција *findOne()* која враћа једног корисника који одговара услову задатом у аргументу (да му је *e-mail* исти као *e-mail­* који је корисник унео у форми за пријављивање).

Уколико функција не нађе ниједног таквог корисника, као одговор на захтев се враћа порука “*Authentification failed*”, са статус кодом 401, што говори кориснику да је дошло до грешке са његове стране. Битно је напоменути да није добра пракса слати кориснику детаљније поруке од овога, јер би теоретски тако могао да извуче информације о постојећим корисницима (не треба му говорити да ли је погрешио код мејла или код лозинке, јер би онда знао да ли постоји корисник са таквим мејлом).

Ако је нађен такав корисник користи се функција *compare()* из *bcrypt.js* библиотеке да упореди унету лозинку са хеш вредношћу лозинке из базе података.

Ако је лозинка тачна прави се објекат *token* уз помоћ *jsonwebtoken* библиотеке.

Затим се направљени токен шаље кориснику заједно са другим релевантним подацима о том кориснику са успешним статус кодом 200.

### Middleware функције

У овом пројекту коришћене су две *middleware* функције: *authenticate* и *authorize*.

*Authenticate* се позива на свакој рути која је заштићена, односно захтева да корисник буде пријављен.

const jwt = require("jsonwebtoken");

const User = require("../models/user");

module.exports = async (req, res, next) => {

  try {

    const token = req.headers.authorization.split(" ")[1];

    const decoded = jwt.verify(token, process.env.JWT\_KEY);

    req.user = await User.findById(decoded.\_id);

    next();

  } catch (error) {

    return res.status(401).json({

      message: "Auth Failed",

    });

  }

};

Из заглавља *HTTP* захтева се извлачи кодирани токен (користи се *split(“ ”)* да би се токен одвојио од стринга „*Bearer“*). Кодирани токен се затим декодира са истом тајном која коришћена за његово кодирање. Затим се проналази корисник чији је то токен (његов јединствени идентификатор је смештен у токену) и тај корисник се смешта у *req.user* варијаблу, којој могу приступити све следеће функције које се извршавају при овом захтеву. На крају се позива функција *next()* која дозвољава да се извршава следећа функција дефинисана у рутеру.

*Authorize* је једноставна функција која се позива на свакој рути која захтева да корисник буде администратор. Изгледа овако:

module.exports = (role) => (req, res, next) => {

  // checks if the user has the permisions to use resource

  if (req.user.role !== role) {

    res.status(401).json("Not authorized");

  }

  next();

};

Проверава да ли је улога корисника та која треба да буде ( у овом пројекту постоје само улоге администратора и корисника, али је остављена могућност да се додају додатне улоге уколико у будућности постоји таква потреба).

## Клијентска апликација

### *Vuetify* библиотека компоненти

### Рутирање

Руте коришћене у овом пројекту су следеће:

1. „/“ - Навигацијом на ову руту кориснику се приказује *Product List Page*, односно страница са приказом свих стрипова које корисник може додати у своју корпу.
2. „/user“ - Све руте које почињу овако су „заштићене“ руте којима корисник може приступити само након што се пријавио на свој кориснички налог (у супротном корисник се „редиректује“ на „/“ руту, то су следеће руте:
   1. „cart“ – Ова рута представља страницу „корпе“ корисника. Овде корисник може видети, или изменити, све стрипове које је додао у своју „корпу“, касније може поручити стрипове који му се налазе у корпи, након чега се корпа празни, како би могао направити нову поруџбину.
   2. „checkout“ – Ова рута представља страницу за прављење поруџбине. Овде корисник уноси своје податке (име, презиме, адресу итд.). Корисник има опцију да сачува ове податке да не би морао да поново уноси ове податке следећи пут.
   3. „orders“ – Ова рута кориснику приказује страницу на којој може видети све поруџбине које је направио и њихово стање („послато“ / „није послато“)
3. „/login“ – Ова рута кориснику приказује страницу за пријаву уколико већ има регистрован налог. У случају да нема постоји дугме које га води до странице за регистрацију.
4. „/register“ – Ова рута приказује страницу за регистровање нових корисника. Корисник овде уноси валидан *e-mail* и своју одабрану лозинку. Уколико не постоји већ корисник са тим *e-mail*-ом, биће успешно регистрован и улогован.
5. „/products/:id“ – Ова рута приказује *Product Details Page*, то јест детаљне податке о одабраном стрипу (*id* је јединствени идентификатор сваког стрипа).
6. „/admin“ - Све руте које почињу овако су заштићене руте којима може приступити само корисник који је администратор, уколико корисник није улогован биће „редиректован“ на „/login“ руту, а ако јесте улогован, али није администратор биће „редиректован“ на почетну страницу. То су следеће руте:
   1. „add-product“ – Ова рута приказује администратору страницу са формом за додавање новог стрипа.
   2. „products“ – Ова рута приказује администратору страницу са компактном листом свих производа (ради лакшег прегледа), са дугметом за брисање и промену поред сваког производа.
   3. „orders“ – Ова рута приказује администратору страницу са свим поруџбинама где може променити статус поруџбина које је послао.
7. „/:catchAll(.\*)“ – На крају, уколико корисник покуша да унесе не постојећу руту, биће му приказана страница која му говори да тражена рута не постоји и даће му се опција да оде на почетну руту.

Пре сваке промене руте имплементирана је функција која проверава да ли корисник сме да приступи тој рути, она изгледа овако:

router.beforeEach((to, from, next) => {

  const userStore = useUserStore();

  if (to.meta.auth && !userStore.token) {

    //auth required

    console.log("Auth required, token: ", userStore.token);

    next("/login");

  }

  if (to.meta.admin && userStore.role !== "admin") {

    //admin required

    console.log("userStore.role: ", userStore);

    next("/");

  }

  next();

});

### Приказ страница

Главни *App.vue* фајл садржи *TheHeader.vue* компоненту која садржи заглавље и мени на извлачење са десне стране који служи за навигацију.

Испод заглавља налази се *router-view* компонента у којој се рендерују жељене компоненте уз помоћ *Vue* рутера.

На дну се налази *TheFooter.vue* која садржи подножје странице у којој су линкови за социјалне мреже.

Шаблон *App.vue* фајла изгледа овако:

<template>

  <v-app theme="dark" class="app">

    <the-header></the-header>

    <v-main>

      <v-container>

        <v-card class="py-5" elevation="3">

          <v-scroll-y-transition :hide-on-leave="true">

            <router-view :key="$route.fullPath"></router-view>

          </v-scroll-y-transition>

        </v-card>

      </v-container>

    </v-main>

    <the-footer></the-footer>

  </v-app>

</template>

*v-scroll-y-transition* је *Vuetify* компонента која примењује ефекте *scroll* анимације приликом мењања рендероване компоненте у *router-view* компоненти.

Као што се види, апликација има само једну страницу (*App.vue*) у којој се мења рендерована компонента и тиме се даје привид постојања више страница. То је принцип рада једностраних апликација (*Single Page Application*).

Пример компоненте која се може рендеровати у *router-view* компоненти:

<template>

  <div>

    <h1 class="text-center text-h3">Error 404: Page not found!</h1>

    <p>

      The page you are looking for could not be found. Retrun to the

      <router-link to="/">Home</router-link> page?

    </p>

  </div>

</template>

Ово је шаблон *NotFoundPage.vue* копмоненте. Уколико корисник унесе непостојећу руту, овај шаблон ће у целости бити рендерован на месту *router-view* компоненте.

### Складиштење податка

У овом пројекту коришћена је *Pinia* за кеширање (*caching*)података добијених од сервера. Коришћена су четири *Pinia* складишта:

* *cart.js*
* *orders.js*
* *products.js*
* *user.js*

Свако од ових складишта садржи „стања података“ (*states*), што су варијабле у које се уписују подаци добијени од сервера.

Свако од ових складишта има своје „акције“ (*actions*)*,* то јест функције које шаљу *HTTP* захтев серверу за ту одговарајућу акцију (нпр. акција fetchProducts() шаље *HTTP GET* захтев серверу, који затим враћа све производе у бази података).

Те акције се позивају у остатку апликације када год је то потребно (нпр. када корисник кликне дугме „додај у корпу“ позива се *addToCart()* акција и онда сервер ажурира стање корпе).

* Пример акције *addToCart()* у *cart.js* складишту:

 async addToCart(productId, quantity) {

      const userStore = useUserStore();

      const headers = {

        Authorization: "Bearer " + userStore.token,

      };

      const reqBody = {

        productId,

        quantity,

      };

      try {

        const res = await axios.post(

          config.API\_URL + "/api/v1/user/cart",

          reqBody,

          {

            headers,

          }

        );

        console.log(res.data);

        console.log(this.items);

        await this.getCart();

      } catch (error) {

        return error.message;

      }

    },

Као што се види у коду, прво се инстацира *userStore* складиште, како би се могло приступити корисниковом *JWT* токену, затим се тај токен ставља у заглавље *HTTP* захтева.

У тело позива се стављају идентификатор жељеног стрипа и количина која се додаје у „корпу“.

Након тога серверу се шаље *HTTP POST* захтев уз помоћ *Axios*-a и ако сервер одговори са успешном поруком, то значи да је производ додат у корпу и позива се акција *getCart()* за *referesh*-овање корпе, да би се у *items* стањe ставила нова листа стрипова у корпи. У супротном акција враћа *Error* поруку коју јој је сервер послао.

* Пример акције *login* у *user.js* складишту:
* async login(email, password) {
* if (this.token) {
* this.logout();
* }
* try {
* const res = await axios.post(config.API\_URL + "/api/v1/user/login", {
* email,
* password,
* });
* this.token = res.data.token;
* this.\_id = res.data.\_id;
* this.name = res.data.name;
* this.email = res.data.email;
* this.role = res.data.role;
* this.checkoutInfo = res.data.checkoutInfo;
* // Persist login state
* await localStorage.setItem(
* "loggedInUser",
* JSON.stringify({
* token: this.token,
* \_id: this.\_id,
* name: this.name,
* role: this.role,
* checkoutInfo: this.checkoutInfo,
* })
* );
* } catch (error) {
* console.log(error);
* }
* },

Акција *login()* узима два аргумента: *email* и *password.*

Уколико је неки претходни корисник већ улогован (ако вредност стања *token* није *null*), позива се акција *logout()*, како би се он одјавио. Затим се шаље *POST* захтев серверу са *email-ом* и *password*-ом.

Затим се подаци добијени од севера уписују у одговарајућа стања у складишту.

И на крају се ови подаци уписују у *localStorage*, на овај начин корисник остаје улогован, чак иако угаси претраживач.

Слика 2.1 – Hhhhhhhhhhhhhhh jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj.

# ЗАКЉУЧАК

Модерне веб апликације су скуп разних и различитих технологија које заједно раде да пружају услуге корисницима.

Прошли су дани када су веб сајтови били сачињени од једноставних статичних *HTML* страница. Данашње потребе тржишта захтевају од девелопера модерних веб апликација да разумеју и баратају широким спектром технологија.

# ИНДЕКС ПОЈМОВА

# ЛИТЕРАТУРА

[1] Б. Раковић, *Електроника – Линеарна интегрисана кола*, Грађевинска књига, Београд, 1983.

[2]D. Davis, E. Patronis Jr, *Sound system Engineering,*pp. 202-212, Focal Press, 2006.

[3] S.H. Linkwits, "Active Crossover Networks for Noncoincident Drivers", *J. Audio Eng. Soc.*, vol. 24, no. 1, pp. 2-8, 1976.

[4]J.J. Lopez, B. Pueo and M. Cobos, “Conventional and distributed mode loudspeaker arrays for the application of wave-field synthesis to video conference” *124th Conv. Audio Eng. Soc.*,Amsterdam, May 2008.

[5] J. Murray, „A Primer On Ethernet Cabling For Digital Audio” <http://www.prosoundweb.com/article/a_primer_on_ethernet_cabling_for_digital_audio/>, преузето: април 2016.

# Прилози

# изјава о академскoј честитости

**ИЗЈАВА О АКАДЕМСКОЈ ЧЕСТИТОСТИ**

|  |  |
| --- | --- |
| **Студент (име, име једног родитеља и презиме):** |  |
| **Број индекса:** |  |

Под пуном моралном, материјалном, дисциплинском и кривичном одговорношћу изјављујем да је завршни рад, под насловом:

1. резултат сопственог истраживачког рада;
2. да овaj рад, ни у целини, нити у деловима, нисам пријављиво/ла на другим високошколским установама;
3. да нисам повредио/ла ауторска права, нити злоупотребио/ла интелектуалну својину других лица;
4. да сам рад и мишљења других аутора које сам користио/ла у овом раду назначио/ла или цитирао/ла у складу са Упутством;
5. да су сви радови и мишљења других аутора наведени у списку литературе/референци који је саставни део овог рада, пописани у складу са Упутством;
6. да сам свестан/свесна да је плагијат коришћење туђих радова у било ком облику (као цитата, прафраза, слика, табела, дијаграма, дизајна, планова, фотографија, филма, музике, формула, вебсајтова, компјутерских програма и сл.) без навођења аутора или представљање туђих ауторских дела као мојих, кажњиво по закону (Закон о ауторском и сродним правима), као и других закона и одговарајућих аката Високе школе електротехнике и рачунарства струковних студија у Београду;
7. да је електронска верзија овог рада идентична штампаном примерку овог рада и да пристајем на његово објављивање под условима прописаним актима Високе школе електротехнике и рачунарства струковних студија у Београду;
8. да сам свестан/свесна последица уколико се докаже да је овај рад плагијат.

У Београду, \_\_. \_\_. 201\_. године

Својеручни потпис студента

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_